

ECE-464: Databases Project Report

Jacob Khalili & Lizelle Ocfemia

[GitHub](#)

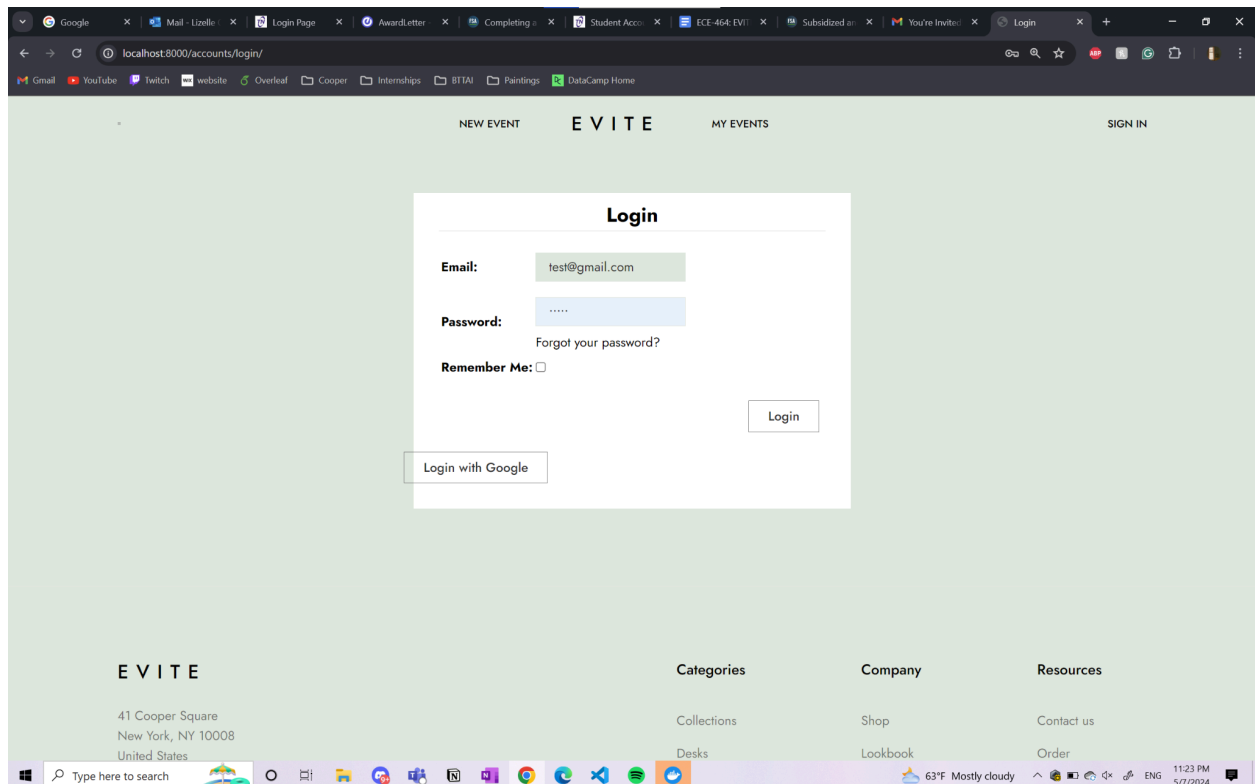
Evite

Planning a party? Hosting an event? Try EVITE: an online invitation maker for every kind of event! Send beautiful, customized invitations with RSVP tracking via email or a shareable link!

Project Overview

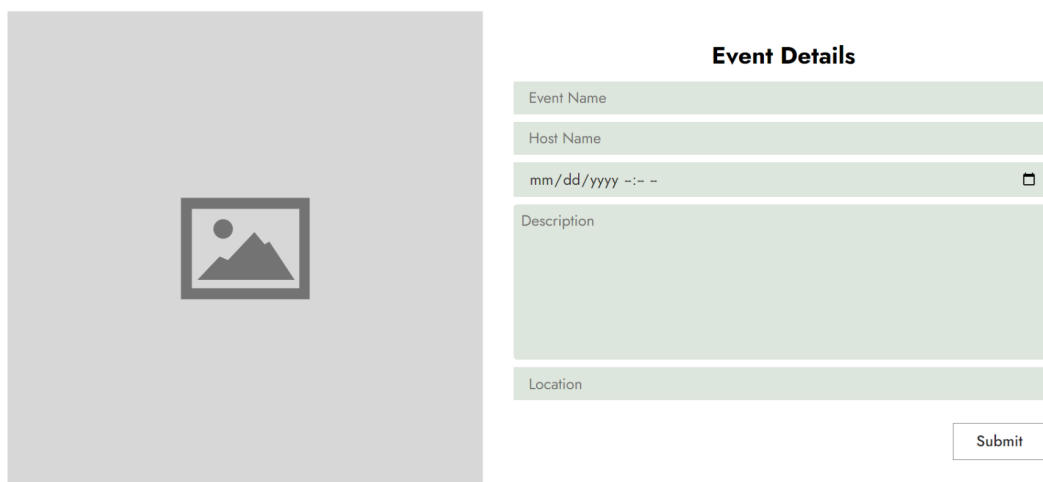
Front-end

Before using Evite, the user must first login with their google account:



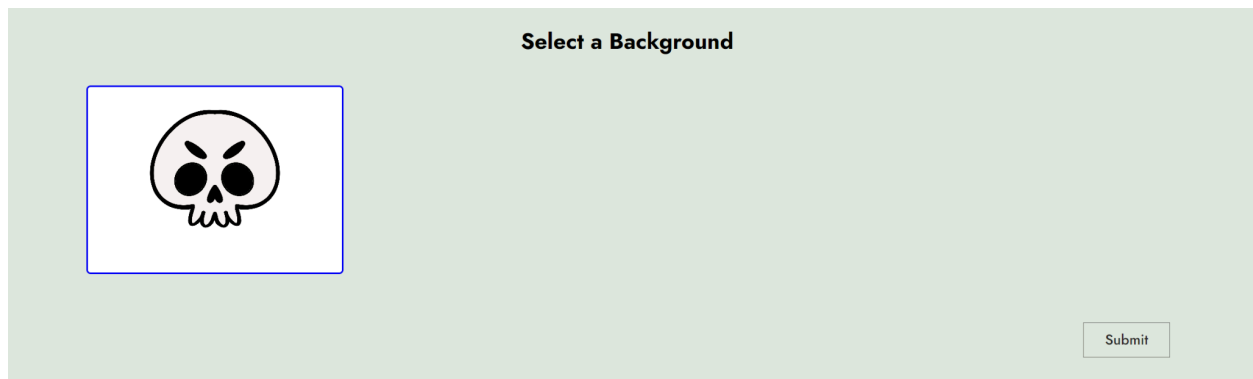
The homepage welcomes the user to create their own invitation for whatever event they have in mind. First, they must upload a scan of their own 5x7 invitation card. This card will be stored as an ImageField in the Invitations model.

Once the user submits his/her card, he/she is directed to edit the Event Details. The form will be inserted into the Event model (title, host, date_time, description, location, respectively).



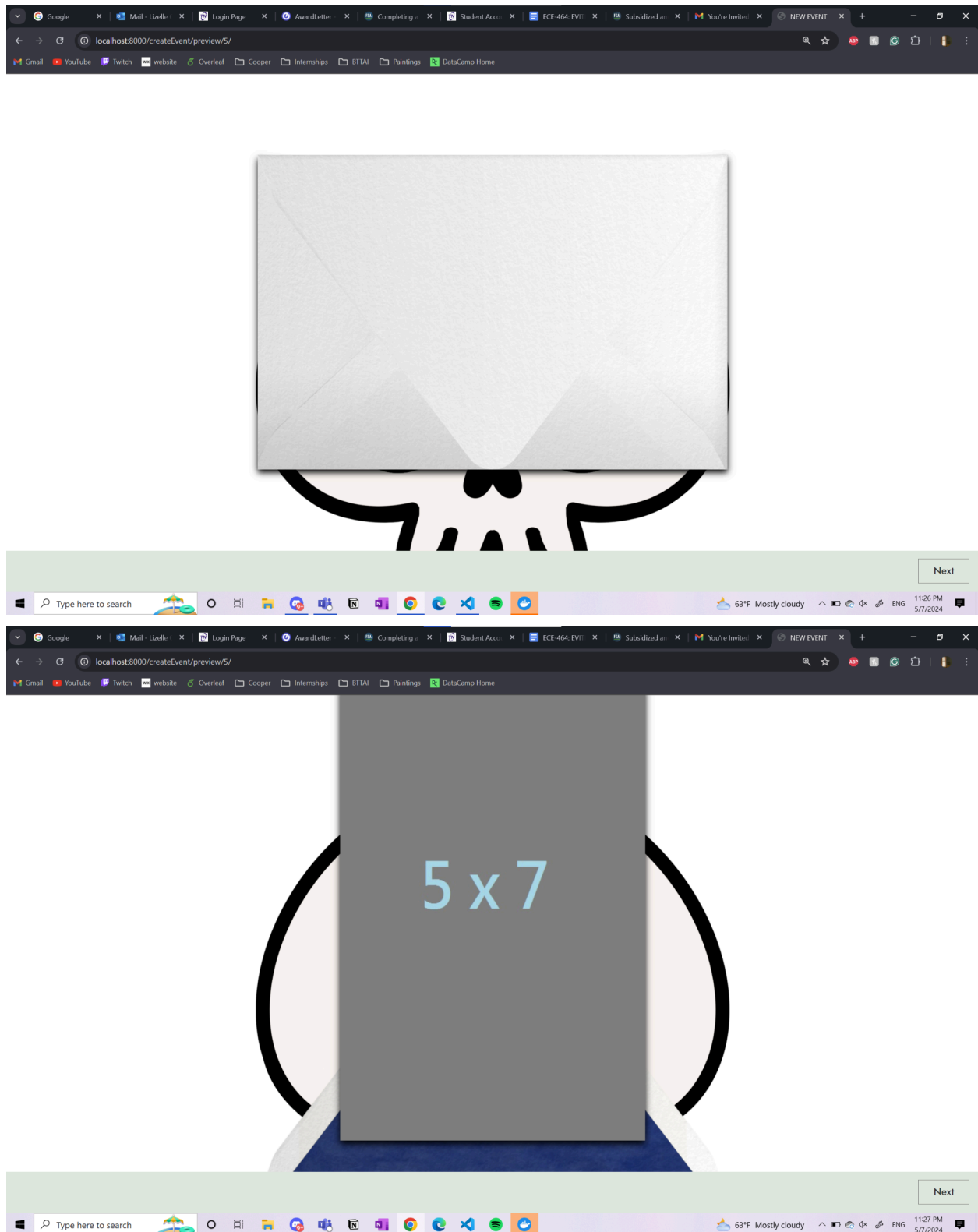
The image shows a web form titled "Event Details" on a light green background. To the left of the form is a large gray square containing a small icon of a picture frame with a mountain and sun. The form consists of several input fields: "Event Name", "Host Name", a date field with the placeholder "mm/dd/yyyy --" and a calendar icon, a large "Description" text area, and a "Location" field. A "Submit" button is located at the bottom right of the form.

The user must select a background image after creating all the required event details. As admins, we can decide what backgrounds the user is allowed to choose from in the Backgrounds model.



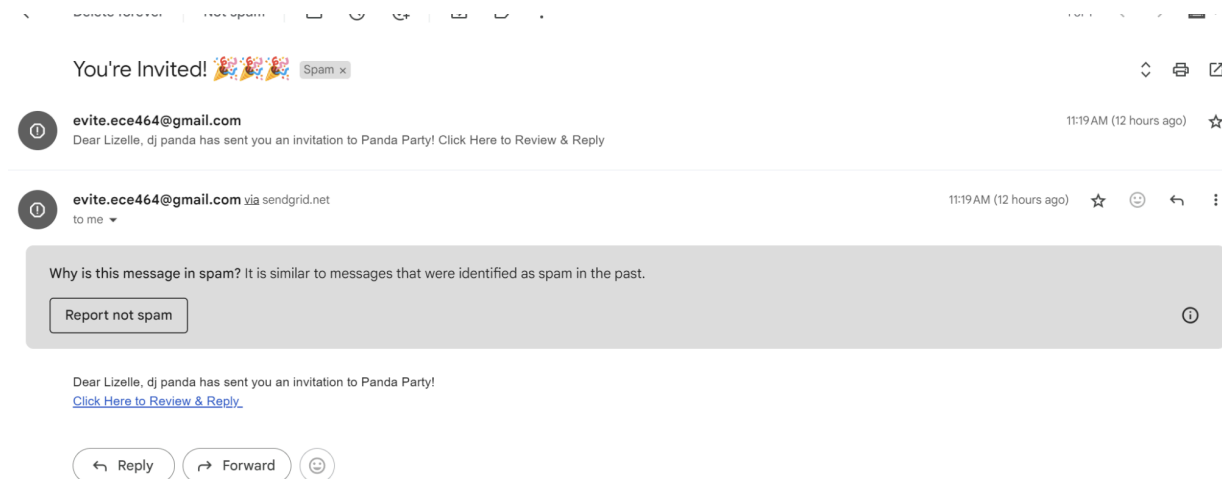
The image shows a web form titled "Select a Background" on a light green background. On the left side of the form is a square image of a skull with a blue border. A "Submit" button is located at the bottom right of the form.

Upon clicking submit, the user is shown a preview of how the invitation animation would look like on the receiver end:

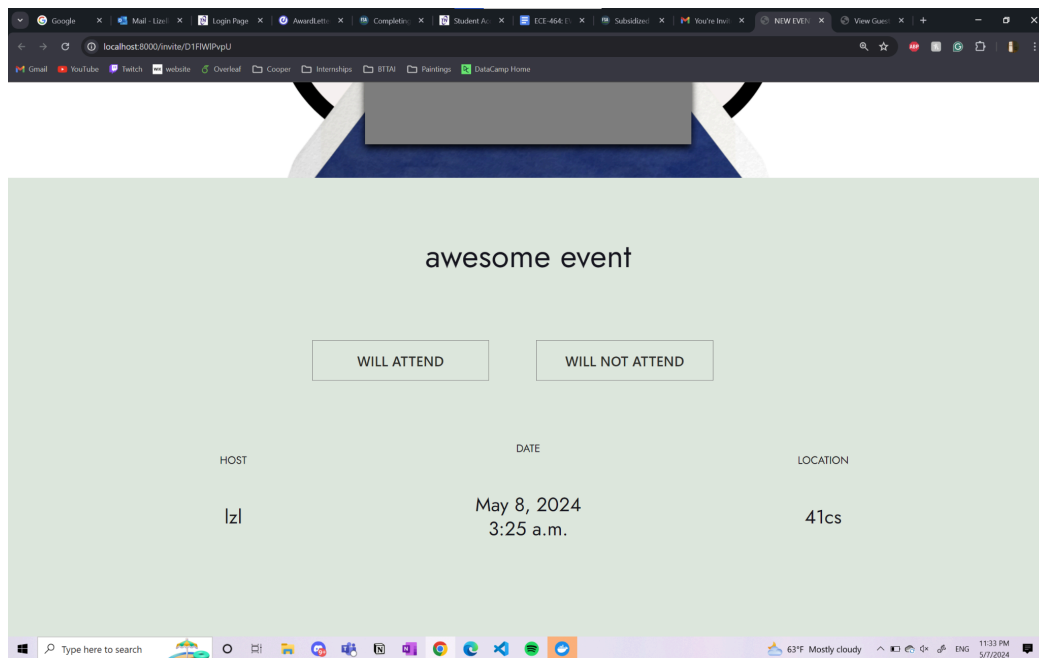


After clicking “Next”, then user is sent to the /editGuests page to add their guests for the event.

Upon clicking “Send!” the receiver is greeted with an email in their Spam folder and a personalized invitation for them to click on.



The link will send them to the final invitation animation along with the details of the event:



Lastly, the /myEvents page will list all the events that the user has created along with the title and description of each event.



Back-end

We have chosen Django for our project due to its great handling of the diverse data types we need to manage. Django simplifies the integration of various data elements, such as incorporating invitations directly into the database via ImageFields. Additionally, one key feature of our application is the ability to create customized backgrounds for these invitations. To ensure these backgrounds remain exclusive and are not user-imported, we've utilized Django's FileField to manage them securely. Below are the detailed models we've implemented:

```
1  from django.db import models
2  from django.conf import settings
3  from django.contrib.auth.models import User
4  from django.core.mail import send_mass_mail, send_mail
5  from .events import Event
6  from .backgrounds import Background
7  from .envelopes import Envelope
8
9  class Background(models.Model):
10     background_names = models.CharField(max_length=500)
11     file = models.ImageField(upload_to='backgrounds/')
12
13     # add in initial backgrounds from admin panel on create
14
15  class Envelope(models.Model):
16     title_env = models.CharField(max_length=50)
17     pattern_env = models.FileField(path="envelopes") # add imgs to envelopes dir
18
19
20  class Invitation(models.Model):
21     event = models.ForeignKey(Event, on_delete=models.CASCADE)
22     background = models.ForeignKey(Background, on_delete=models.CASCADE, null=True)
23     envelope = models.ForeignKey(Envelope, on_delete=models.CASCADE, null=True)
24     card = models.ImageField(upload_to='cards/', null=True, blank=True)
25
26
27  class Event(models.Model):
28     title = models.CharField(max_length=200)
29     description = models.TextField()
30     location = models.CharField(max_length=200)
31     date_time = models.DateTimeField(auto_now_add=True, null=True)
32     owner = models.ForeignKey(User, on_delete=models.CASCADE)
33     host = models.TextField(null=True)
34
35  class Guest(models.Model):
36
37     event = models.ForeignKey(Event, on_delete=models.CASCADE)
38     name = models.CharField(max_length=50, null=True)
39     email = models.EmailField()
40
41     phone = models.CharField(max_length=11, null=True, blank=True)
42     status = models.CharField(null=False, default='Not Sent', max_length=20, choices=STATUS_CHOICES)
43     hash = models.CharField(
44         max_length=10,
45         default=create_new_hash,
46         unique=True,
47         editable=False
48     )
49
```

Django Models

The models outlined below are part of a Django application designed to manage invitations and events. Each model serves a specific function within the system, working together to handle everything from user data to media storage.

Background Model:

The Background model is designed to store image files that serve as backgrounds for the invitations. Each background has a name (`background_names`) and is stored as an image file in the designated directory (`'backgrounds/'`). Administrators can add initial background options through the Django admin panel.

Envelope Model:

The Envelope model represents different envelope designs available for the invitations. It includes a title (`title_env`) and a file path (`pattern_env`) that points to images stored in the `'envelopes'` directory. These images represent the various envelope patterns users can choose from.

Invitation Model:

The Invitation model is central to the application, linking invitations to specific events, backgrounds, envelopes, and optional card images. It uses foreign keys to associate an invitation with a particular event (`event`), background (`background`), and envelope (`envelope`). There is also an option to include a separate card image (`card`), which is uploaded to a specified directory (`'cards/'`).

Event Model:

The Event model captures details about events for which invitations are sent. It stores the event's title, description, location, and timing (`date_time`). Additionally, it includes references to the User model to identify the event's owner and an optional text field for the host's details. The creation time of the event is automatically set upon record creation. The

Event model also has its own `sendInvites()` command, which sends each of its guests their own personal email with an invite link.

Guest Model:

The Guest model maintains information about the invitees to an event, including their name, email, and optionally their phone number. It also tracks the invitation status (e.g., Not Sent, Sent, Confirmed) through the status field, which is linked to predefined choices ("status_choices"). Each guest record is associated with an event and includes a unique hash (hash), generated by the function `create_new_hash`, to uniquely identify the guest record.

Django URLs

With Django, it is much easier to create different paths for the user to move to. The following code snippet defines the URL patterns for routing requests within a Django application.

```
urlpatterns = [
    # path('', index.index, name='index'),
    path('', event_views.createEvent, name='index'),
    path('myEvents/', event_views.myEvents, name='myEvents'),
    path('createEvent/', event_views.createEvent, name='createEvent'),
    path('createEvent/background/', event_views.selectBackground, name='selectBackground'),
    path('createEvent/preview/<int:event_id>', event_views.eventPreview, name='eventPreview'),
    path('createEvent/guests/<int:event_id>', event_views.guestPage, name='addGuests'),

    path('editGuests/', event_views.editGuests, name='editGuests'),
    path('newGuest/', event_views.newGuest, name='newGuest'),
    path('editGuests/<str:hash>', event_views.editGuests, name='editGuests'),

    path('createEvent/guests/updateRow/', index.updateRow, name='updateRow'),
    path('createEvent/guests/table/<int:event_id>', index.getTableData, name='deleteRow'),
    path('getStats/<int:event_id>', index.getStats, name='getStats'),

    path('invite/event_id/<int:event_id>', event_views.get_animation, name='preview-invite'),
    path('invite/<str:hash>', event_views.getInvitePage, name='invite'),

    # send invitation(s)
    path('sendInvitation/<int:event_id>/<int:guest_id>', event_views.sendOneInvitation, name='send-invitation'),
    path('sendAllInvitations/<int:event_id>', event_views.sendAllInvitations, name='send-all-invitations'),
]
```

Each route is paired with a corresponding view function to handle incoming requests. For example, paths such as the home page `/` are directed to functions like `event_views.createEvent`, which renders the page for creating new events. Other paths like `'myEvents/'` and `'createEvent/guests/'`

<int:event_id>/' route to functions such as event_views.myEvents and event_views.guestPage, respectively, to manage specific functionalities like displaying a user's events or adding guests to an event. Additionally, the code includes routes for handling invitation-related actions, such as sending invitations individually or in bulk. Overall, these URL patterns organize the application's endpoints and define how incoming requests are processed and responded to by the server.

Sendgrid

We have implemented our email-sending functionality with the use of Sendgrid. Instead of allowing the user to email each of his/her guests from his/her own personal email, we created a new email (evite.ece464@gmail.com) that all invitations will be sent by. Unfortunately, the emails have been directed to the spam inbox of the receiver because we are not using our own domain.